A Secure Submission System for Online Whistleblowing Platforms

Volker Roth, Benjamin Güldenring, Eleanor Rieffel,† Sven Dietrich,‡ Lars Ries

Freie Universität Berlin, †FX Palo Alto Laboratory,
‡Stevens Institute of Technology

January 29, 2013

Abstract

Whistleblower laws protect individuals who inform the public or an authority about governmental or corporate misconduct. Despite these laws, whistleblowers frequently risk reprisals and sites such as WikiLeaks emerged to provide a level of anonymity to these individuals. However, as countries increase their level of network surveillance and Internet protocol data retention, the mere act of using anonymizing software such as Tor, or accessing a whistleblowing website through an SSL channel might be incriminating enough to lead to investigations and repercussions. As an alternative submission system we propose an online advertising network called *AdLeaks*. AdLeaks leverages the ubiquity of unsolicited online advertising to provide complete sender unobservability when submitting disclosures. AdLeaks ads compute a random function in a browser and submit the outcome to the AdLeaks infrastructure. Such a whistleblower's browser replaces the output with encrypted information so that the transmission is indistinguishable from that of a regular browser. Its back-end design assures that AdLeaks must process only a fraction of the resulting traffic in order to receive disclosures with high probability. We describe the design of AdLeaks and evaluate its performance through analysis and experimentation.

1 Introduction

Corporate or official corruption and malfeasance can be difficult to uncover without information provided by insiders, so-called *whistleblowers*. Even though many countries have enacted, or intend to enact, laws meant to make it safe for whistleblowers to disclose misconduct [3, 28], whistleblowers fear discrimination and retaliatory action regardless, and sometimes justifiably so [11, 26].

It is therefore unsurprising that whistleblowers often prefer to blow the whistle anonymously through other channels than those mandated by whistleblowing legislature. This gave rise to whistleblowing websites such as *Wikileaks*. However, the proliferation of surveillance technology and the retention of Internet protocol data records [4] has a chilling effect on potential whistleblowers. The mere act of connecting to a pertinent Website may suffice to raise suspicion [20], leading to cautionary advice for potential whistleblowers.

The current best practice for online submissions is to use an SSL [19] connection over an anonymizing network such as Tor [17]. This hides the end points of the connection and it protects against malicious exit nodes and Internet Service Providers (ISPs) who may otherwise eavesdrop on or tamper with the connection. However, this does not protect against an adversary who can see most of the traffic in a network [12, 18], such as national intelligence agencies with a global reach and view.

In this paper, we suggest a submission system for online whistleblowing platforms that we call *AdLeaks*. The objective of AdLeaks is to make whistleblower submissions unobservable even if the adversary sees the entire network traffic. A crucial aspect of the AdLeaks design is that it eliminates any signal of intent that could be interpreted as the desire to contact an online whistleblowing platform. AdLeaks is essentially an online advertising network, except that ads carry additional code that encrypts a zero probabilistically with the AdLeaks public key and sends the ciphertext back to AdLeaks. A whistleblower's browser substitutes the ciphertext with encrypted parts of a disclosure. The protocol ensures that an adversary who can eavesdrop on the network communication cannot distinguish between the transmissions of regular browsers and those of whistleblowers' browsers. Ads are digitally signed so that a whistleblower's browser can tell them apart from maliciously injected code. Since ads are ubiquitous and there is no opt-in, whistleblowers never have to navigate to a particular site to communicate with AdLeaks and they remain unobservable. Nodes in the AdLeaks network reduce the resulting traffic by means of an *aggregation* process. We designed the aggregation scheme so that a small number of trusted nodes with access to the decryption keys can

recover whistleblowers' submissions with high probability from the aggregated traffic. Since neither transmissions nor the network structure of AdLeaks bear information on who a whistleblower is, the AdLeaks submission system is immune to passive adversaries who have a complete view of the network.

In what follows, we detail our threat model and our assumptions, we give an overview over the design of AdLeaks, we analyze its scalability in detail, we report on the current state of its implementation and we explain how AdLeaks uses cryptographic algorithms to achieve its security objectives.

2 Threat Model

The primary security objective of AdLeaks is to conceal the *presence* of whistleblowers, and to eliminate network traces that may make one suspect more likely than another in a search for a whistleblower. This security objective is more important to AdLeaks than, for example, availability. We rather risk that a disclosure does not come through than compromise information about a whistleblower. In what follows we detail the threats our system architecture addresses as well as the threats it does not address.

2.1 Threats in Our Scope

AdLeaks addresses the threat of an adversary who has a global view of the network and the capacity to store or obtain Internet protocol data records for most communications. The adversary may even require anonymity services to retain connection detail records for some time and to provide them on request. The adversary may additionally store selected Internet traffic and he may attempt to mark or modify communicated data. However, we assume that the adversary has no control over users' end hosts, and he does not block Internet traffic or seizes computer equipment without a court order. We assume that the court does not *per se* consider organizations that relay secrets between whistleblowers and journalists as criminal. The objective of the adversary is to uncover the identities of whistleblowers. The threat model we portrayed is an extension of [4] and it is likely already a reality in many modern states, or it is about to become a reality. For reasons we explain in the following section we do not consider additional threats that we would doubtless encounter, for example, in technologically advanced totalitarian countries.

2.2 Threats Not in Our Scope

We exclude blocking from our threat model and our discussion because we do not contribute to blocking resistance and its inclusion would distract from our contribution. The second threat we exclude is that of a flooding attack on our submission system. While we have thoughts on how to limit some attacks of this kind we prefer to make a solid first step towards unobservability before considering the next step in our research. We hope that the next step will not become necessary because this means that countries we believe liberal have gone too far down the slope towards totalitarianism already. The third threat we exclude is denial of service by means of fake transmissions. This threat manifests at the level of the editorial process that separates the chaff from the wheat among the potentially many submitted disclosures. We consider this threat out of scope in this part of our work. The fourth threat we exclude is that of malware and spyware. For example, sensitive documents in PDF format may contain JavaScript that emits a beacon whenever the document is viewed. A careless whistleblower who opens a sensitive document on his home machine may expose himself or herself in that fashion [5]. Similarly, if a whistleblower's computer is infected by a malware or spyware then the whistleblower has no security.

2.3 Security-Related Assumptions

AdLeaks ads require a source of randomness in the browser that is suitable for cryptographic use. Moreover, the source must be equally good on regular browsers and on the browsers of whistleblowers. If a whistleblower's browser looks decidedly more random than other browsers then whistleblowers can be readily identified. Unfortunately the random numbers most browsers generate are far from random. However, there are good reasons for browser developers to support cryptographically secure random number generators in the near-term [15], for example, to prevent illicit user tracking [25]. In the meantime, entropy collected in the browser may be folded into a pseudorandom generator using SJCL [34]. We therefore decided to move forward with our research assuming that browsers will soon be ready for it.

We assume that whistleblowers use AdLeaks only on private machines to which employers have no access. In fact, sending information from work computers even using work-related e-mail accounts is a mistake whistleblowers make frequently. We hope that the software distribution channels we discuss in Section 3.3 will help reminding whistleblowers to not make that mistake.

3 System Architecture

AdLeaks consists of two major components. The first component is an online advertising network compara-

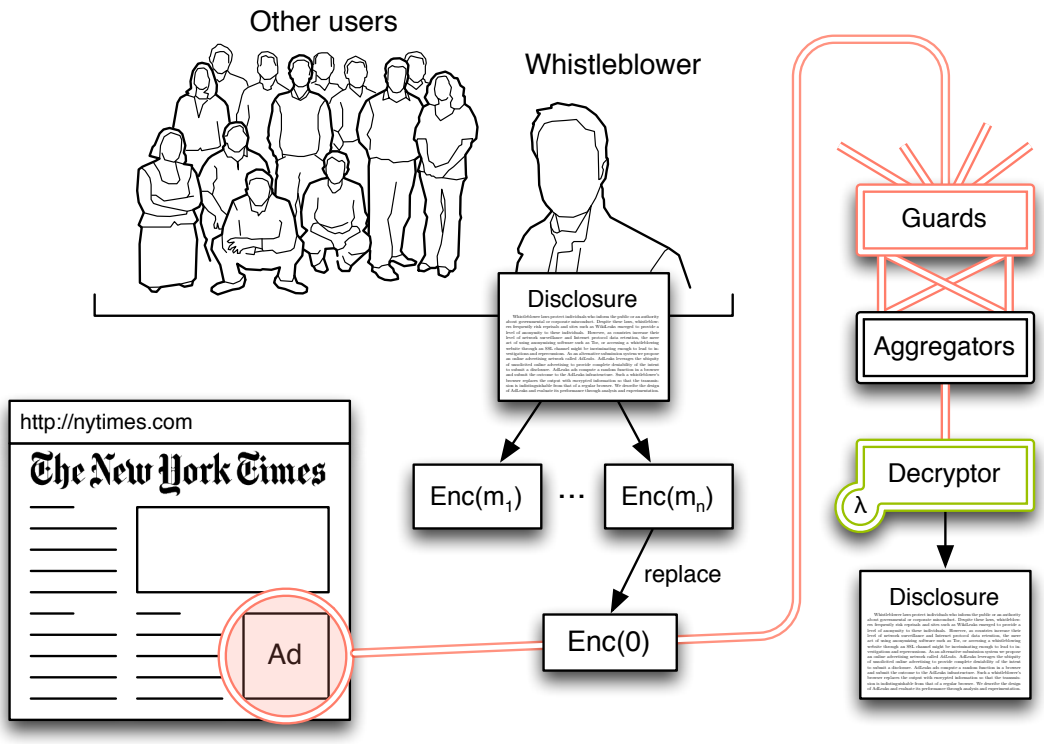

Figure 1: Illustrates the architecture of the AdLeaks system. Aggregators reduce the incoming traffic so that submissions can be funneled to a trusted decryptor through a household DSL line. λ denotes the decryption key.

ble to existing ones. The network has advertising partners (the publishers) who include links or scripts in their web pages which request ads from the AdLeaks network and display them. Publishers may receive compensation in accordance with common advertising models, for example, per mille impressions, per click or per lead generated. Advertisers run campaigns through the AdLeaks network. AdLeaks may additionally run campaigns through other ad networks to extend its reach, for example, funded by donations or profits from its own operations. The ecosystem of partners and supporters may include large newspapers, bloggers, human rights organizations and their affiliates. For example, Wikileaks has partnered with organizations such as Der Spiegel, El País and the New York Times, and OpenLeaks had hinted at support by Greenpeace and other organizations. The key ingredient of an AdLeaks ad is not its visual display but its active JavaScript content. Supporters who would forfeit significant revenue when allocating advertising space to AdLeaks ads have a choice to only embed the JavaScript portion. The JavaScript is digitally signed by AdLeaks and contains public encryption keys.

The second major component of AdLeaks is its submission infrastructure. This infrastructure consists of three tiers of servers. We refer to these tiers as *guards, aggregators* and *decryptors.* When a browser loads an AdLeaks ad, the embedded JavaScript encrypts a zero probabilistically with the embedded public key and submits the ciphertext to a guard. The guard strips unnecessary encoding and protocol meta-data from the request and forwards the ciphertext to an aggregator. An aggregator aggregates the ciphertexts it receives per second and transmits them to the decryptor. What makes this setting challenging is that we want to limit the bandwidth of the decryptor to a household Internet connection so that we can keep a close eye on the all-important machine with the decryption keys. The aggregation leverages the homomorphic properties of the Damgård-Jurik (DJ) encryption scheme [16], which means that the product of the ciphertexts is an encryption of the sum of the plaintexts. We chose the DJ scheme because it has a favorable plaintext to ciphertext ratio.

The decryptor decrypts the downloaded ciphertexts and, if it finds data in them, reassembles the data into files. The files come from whistleblowers. In order to submit a file, a whistleblower must first *obtain* an installer that is digitally signed and distributed by AdLeaks. This is already a sensitive process that signals intent. We defer the discussion of safe distribution channels for the installer to section 3.3. *Installing* the obtained software likewise signals the intent to disclose a secret, and therefore it is crucial that the whistleblower

verifies the signature *before* running the installer, and assures himself that the signer is indeed AdLeaks. Otherwise, he is vulnerable to Trojan Horse software designed to implicate whistleblowers. When run, the installer produces an instrumented browser and an encryption tool. The whistleblower prepares a file for submission by running the encryption tool on it. The tool's output is a sequence of ℓ ciphertexts. Henceforth, whenever an instrumented browser runs an ad signed by AdLeaks, it replaces the script's ciphertext with one of the ℓ ciphertexts it has not already used as a replacement.

In order to distinguish ciphertexts that are encryptions of zeros from ciphertexts that are encryptions of data we refer to the former as *white* and to the latter as *gray*. If the aggregator aggregates a set of white ciphertexts then the outcome is another white one. If exactly one gray ciphertext is aggregated with only white ones then the outcome is gray as well. If we decrypt the outcome then we either recover the data or we determine that there was no data to begin with. If two or more gray ciphertexts are aggregated then we cannot recover the original data from the decryption. We call this event a *collision* and we refer to such an outcome as a *black* one. Obviously, we must expect and cope with collisions in our system. In what follows, we elaborate on details of the design that are necessary to turn the general idea into a feasible and scalable system.

3.1 Disclosure Preparation

In order to handle collisions, the encryption tool breaks a file into blocks of a fixed equal size and encodes them with a loss tolerant *Fountain Code*. Fountain codes encode n packets into an infinite sequence of output packets of the same size such that the original packets can be recovered from any n' of them where n' is only slightly larger than n. For example, a random linear Fountain Code decodes the original packets with probability $1-\delta$ from about $n + \log_2(1/\delta)$ output packets [27]. Let n'' be somewhat larger than n' and let $m_1, \ldots, m_{n''}$ be the Fountain encoding of the file. The tool then generates a random file identification number k and computes: $c_i = \mathrm{Enc}^{\mathrm{cca}}_{\kappa_1}(\mathrm{EncData}_{\kappa_2}(m_i, k||i||n))$ for $1 \leq i \leq n''$ where κ_1 is an aggregator key and κ_2 is the actual submission key. The purpose of the dual encryption will become clear in Section 5.5. We assume that the outer encryption is a fast hybrid IND-CCA secure cipher such as Elliptic Curve El Gamal with AES in OCB mode [32]. We defer the specification of the inner encryption scheme to Section 4. It assures that, when the decryptor receives the ciphertexts, it can verify the integrity of individual chunks and of the message as a whole and he can associate the chunks that belong to the same submission with all but negligible probability (in $|k|$).

3.2 Decryption

It is substantially cheaper to multiply two DJ ciphertexts in the ciphertext group than it is to decrypt one. Furthermore, the product of ciphertexts decrypts to the sum of the plaintexts in the plaintext group. Recollect that we expect to receive a large number of white ciphertexts, that is, encryptions of zeroes. This leads to the following optimization: we form a full binary tree of fixed height, initialize its leaves with received ciphertexts $c_1, \ldots, c_n$ and initialize each inner node with the product of its children. Then, we begin to decrypt at the root. If the plaintext is zero then we are done with this tree, because all nodes in the tree are zeroes. Otherwise, the decryption yields $\gamma = \alpha + \beta \neq 0$ where α, β are the plaintexts of the left and right child, respectively. We decrypt the left child, which yields α, and calculate the plaintext of the right child as $\beta = \gamma - \alpha$ (without explicit decryption). If α or β are zeroes then we ignore the corresponding subtree. Otherwise, we recurse into the subtrees that have non-zero roots. If a node is a leaf then we decrypt and verify it. If we find it invalid then we ignore the leaf. Otherwise we forward its plaintext to the file reassembly process. We quantify the benefits of this algorithm in Section 6.4.

3.3 Software Dissemination

We cannot simply offer the installer software for download because the adversary would be able to observe that. Instead, we pursue a multifaceted approach to software distribution. Our simplest and preferred approach involves the help of partners in the print media business. At the time of writing, popular print media often come with attached CDROMs or DVDs that are loaded with, for example, promotional material, games, films or video documentaries. Our installer software can be bundled with these media. Our second approach is to encode the installer into a number of segments using a Fountain Code. In this approach, AdLeaks ads randomly request a segment that the browser loads into the cache. A small bootstrapper program extracts the segments from the browser cache and decodes the installer from it when enough of them have been obtained. Since extraction happens outside the browser it cannot be observed from within the browser. The bootstrapper can be distributed in the same fashion. This reduces the distribution problem to extracting a specific small file from the cache, for example, by searching for a file with a specific signature or name in the cache directory. This task can probably be automated for most platforms with a few lines of script code. The code can be published periodically by trusted media partners in print or verbatim in webpages or it could even be printed on T-Shirts. Our third approach is to enlist partners who

bundle the bootstrapper with distributions of popular software packages so that many users obtain it along with their regular software. With our multifaceted approach we hope to make our client software available to most potential whistleblowers in a completely innocuous and unobservable fashion.

4 Ciphertext Aggregation

Our ciphertext aggregation scheme is based on the Damgård-Jurik (DJ) scheme, which IND-CPA secure and is also an isomorphism of

$$\psi_s : \mathbb{Z}_{N^s} \times \mathbb{Z}_N^* \leftrightarrow \mathbb{Z}_{N^{s+1}}^*$$
$$\psi_s(a; b) \mapsto (1+N)^a \cdot b^{N^s} \bmod N^{s+1}$$

where N is a suitable public key. The parameter s controls the ratio of plaintext size and ciphertext size. We use two two DJ encryptions c, t to which we jointly refer as a *ciphertext*. We refer to t separately as the *tag*. The motivation for this arrangement is improved performance. We wish to encrypt long plaintexts and the costs of cryptographic operations increase quickly for growing s. Therefore we split the ciphertext into two components. We use a shorter component with $s = 1$, which allows us to test quickly whether the ciphertext encrypts data or a zero. The actual data is encrypted with a longer component with $s > 1$. The two components are glued together using Pederson's commitment scheme [29], which is computationally binding and perfectly hiding. This requires two additions to the public key, which are a generator g of the quadratic residues of $\mathbb{Z}_N^*$ and some $h = g^x$ for a secret x. Instead of committing to a plaintext the sender commits to the hash of the plaintext and some randomness. We use a collision resistant hash function H for this purpose, which outputs bit strings of length $|N/16|$. Furthermore, let R be a source of random bits. The details of the data encryption and decryption algorithms are as follows:

$\text{EncData}(m, r_0) =$
  $r_1, r_2 \leftarrow R$
  $\text{chk} \leftarrow$ if $m, r_0 = 0$ then 0
    else $H(m, r_0)$
  $c \leftarrow \psi(m; h^{\text{chk}} \cdot g^{r_1}))$
  $t \leftarrow \psi(r_0||r_1; g^{r_2})$
  return c, t

$\text{DecVrfy}(c, t) =$
  $(m; k), (r_0||r_1; \cdot) \leftarrow \psi^{-1}(c), \psi^{-1}(t)$
  $\text{chk} \leftarrow$ if $m, r_0 = 0$ then 0
    else $H(m, r_0)$
  if $h^{\text{chk}} \cdot g^{r_1} = k$ then
    return m, r_0
  return $\perp$

We assume that $|r_0|, |r_1|, |r_2|$ are polynomial in the security parameter. Here, r_0 corresponds to $k||i||n$ as we introduced it in Section 3.1. We define EncZero = EncData$(0, 0)$. Aggregation is simply the multiplication of the respective ciphertext components. We establish the correctness of decryption next. Let c, t and c', t' be two ciphertexts. Then

$$\begin{aligned} c \cdot c' &= \psi(m + m'; h^{H(m,r_0)+H(m',r_0')} \cdot g^{r_1+r_1'}) \\ t \cdot t' &= \psi((r_0||r_1) + (r_0'||r_1'); g^{r_2+r_2'}) \\ &= \psi((r_0 + r_0')||(r_1 + r_1'); g^{r_2+r_2'}) \end{aligned}$$

if r_0, r_1, r_0', r_1' are left-padded with zeroes, which we hereby add to the requirements. The amount of padding determines how many ciphertexts we can aggregate in this fashion before an additive field overflows into an adjacent one and corrupts the ciphertext. If we use B bits of padding then we can safely aggregate up to 2^B ciphertexts. A length of $B = 40$ is enough for our purposes. Observe that the aggregation of two outputs of EncData is valid if and only if

$$H(m, r_0) + H(m', r_0') \equiv H(m + m', r_0 + r_0') \quad (1)$$

modulo $\phi(N)$. This amounts to finding a collision in H and we assume that this is infeasible if H resembles a pseudorandom function. On the other hand, if one input to the aggregation is an output of EncData and another input is an output of EncZero then by the definitions of our algorithms we have

$$H(m, r_0) + 0 \equiv H(m + 0, r_0 + 0)$$

modulo $\phi(N)$, which is trivially fulfilled. Hence, a valid data encryption can be modified into another valid encryption of the same data but not into a valid encryption of different data. For this reason our scheme is not IND-CCA secure, although it would achieve the weaker notion of *Replayable CCA* [8] if we removed the special case $m, r_0 = 0$. Unfortunately we need *some* special case to enable aggregation. Therefore we wrap the output of EncData into an IND-CCA secure outer encryption in order to prevent adversaries from collecting samples for use in a bait attack (see Section 5.5).

5 Security Properties

5.1 Traffic Analysis

AdLeaks funnels all incoming transmissions to the decryptor, and transmissions occur without any explicit user interaction. Hence, the posterior probability that anyone is a whistleblower, given his transmission is observed anywhere in the AdLeaks system, equals his prior

probability. From that perspective, AdLeaks is immune against adversaries who have a complete view of the network. Furthermore, AdLeaks' deployment model is suitable to leapfrog the long-drawn-out deployment phase of anonymity systems that rely on explicit adoption. For example, if *Wikipedia* deployed an AdLeaks script then AdLeaks would reach 10% of the Internet user population overnight, based on traffic statistics by *Alexa* [1].

5.2 Denial of Service by Blocking

AdLeaks is vulnerable to blocking. Because anyone should be able to use AdLeaks, anyone must be able to obtain the AdLeaks client software, including the adversary. It is easy to turn the client software into a classifier that learns blocking filters for AdLeaks ads. Furthermore, it is harder in the AdLeaks case to bypass blocking than it is in the case of, for example, censorship resistance software. A whistleblower cannot trust anyone and therefore it is very risky for him to obtain helpful information by gossiping, which works for Tor. Neither can he count on the help of Internet Service Providers, which is the basis for systems such as *Telex* [42], *Cirripede* [21] and *Decoy Routing* [24]. We are not aware of a practical mechanism that is applicable to AdLeaks and that provides satisfactory security guarantees. Therefore, we defer countermeasure design to future research.

5.3 Denial of Service by Flooding

If we deployed one decryption unit and operated at its approximate limit, that is, 51480 concurrent whistleblowers, then it would receive already about 2827 disclosures per day on average. In other words, the editorial backend of AdLeaks would be overwhelmed even before the technical infrastructure is saturated. Under these conditions, the benefit of protecting against flooding attacks is debatable.

5.4 Transmission Tagging

We have shown before that the encryption scheme AdLeaks uses is secure against adaptive chosen ciphertext attacks as long as aggregators are honest. This prevents adversaries from tagging or adding chunks en route to aggregators. The Fountain code ensures that AdLeaks can determine when it has enough chunks to recover the entire disclosure. This prevents any attempts to tag disclsoures by means of dropping or re-ordering chunks.

5.5 Dishonest Aggregators

Assume that AdLeaks did not use outer encryption. Then adversaries might employ the following *active* strategy to gain information on who is sending data to AdLeaks. The adversary samples ciphertexts of suspects from the network and aggregates the ciphertexts for each suspect. He prepares a genuine-looking disclosure that is enticing enough so that the AdLeaks editors will want to publish it with high priority. We call this disclosure the *bait*. The adversary then aggregates suspects' ciphertexts to his disclosure and submits it. If AdLeaks does *not* publish the bait within a reasonable time interval then the adversary concludes that the suspect is a whistleblower. The reasoning is as follows. If the suspect ciphertexts were zeroes then the bait is received and likely published. Since the bait was not published, the suspect ciphertexts carried data which invalidated the bait ciphertexts. This idea can be generalized to an adaptive and equally effective non-adaptive attack that identifies a single whistleblower in a group of W suspects at the expense of $\log_2 W$ baits. For this reason, AdLeaks employs an outer encryption which prevents this attack. However, if an adversary takes over an aggregator then he is again able to launch this attack. Therefore, aggregators should be checked regularly, remote attestation should be employed to make sure that aggregators boot the correct code, and keys should be rolled over regularly. Note that it may take months before a disclosure is published and that a convincing bait has a price — the adversary must leak a sufficiently attractive secret in order to make sure it is published. From this, the adversary only learns that a suspect has sent something but not what was sent.

5.6 Bandwidth-Based Attacks

The adversary may submit a bait while sending a suspect chunk at a rate that is close to or exceeds the data capacity between aggregators and decryptors. If the suspect chunk is a zero then the recovery probability of AdLeaks remains within the expected bounds. If, however, the suspect chunk is a data chunk then, with good probability, AdLeaks does not recover the bait. However, AdLeaks will notice the reduced recovery probability and may react, for example, by rolling over to a new key or even by pushing warnings to whistleblowers via its ad distribution mechanism.

5.7 Client Compromise

If the computer of a whistleblower is compromised then the whistleblower loses all security guarantees. Limited resilience to detection could perhaps be achieved by employing malware-like hiding-tactics. However, since the AdLeaks client is public, it is merely a matter of time until its tactics are reverse engineered and a detection software is written. If the detection software can be

pushed to suspects' computers then whistleblowers can be uncovered. In order to limit the risk, a "production-grade" implementation of the AdLeaks client should offer a function to delete itself securely when it is not needed anymore.

6 Scalability

Our goal in this section is to characterize how well AdLeaks scales with a growing number of users and whistleblowers. Among other dimensions, we explore the necessary and sufficient size of the required infrastructure and the time it takes to submit a file. An estimate of the financial feasibility of the operation can be found in Section 7.

6.1 Submission Duration

In the absence of better data, we analyzed the *Wikileaks* archives available from *wlstorage.net* and estimated the sizes of disclosures as follows: we counted top-level files and archives as individual disclosures; we counted the contents of subdirectories as a single disclosure unless the contents also appeared in an archive. We found that 70% of the disclosure estimates were less than 2 MB (about 20% were larger than 4 MB) and chose 2 MB to be our target size.

Zhang and Zhao conducted a study on tabbed browsing behavior [43] and measured 89,851 page loadings distributed over 20 participants and 31 days, which amounts to about 145 page loadings per user per day. Webpages often display multiple ads. It is common to display a horizontal ad at the top and one or more vertical ads in the margins. The popularity of a website also plays an important role for how many ad loadings it can trigger. News websites elicit frequent and regular page loadings and are particularly suitable for our purposes. Fortunately for us, they also have incentives to support online whistleblowing systems. Furthermore, they might support persistent "ad-less" ads, that is, AdLeaks scripts that do not take up page real estate and therefore do not compete with other ad revenue sources for the news outlet. Lastly, our ads can trigger multiple transmissions at random intervals if we are below our target. For this reason, we decided that it is fair to assume we would get about 50 transmissions per day and user from our ads.

For a sound level of security, we use a public key modulus with 2048 bits. The DJ scheme is expensive for increasing plaintext lengths and based on initial tests we settled on parameters that support 2303 bytes for data use. Our file would thus require about 911 blocks. If we account for the encoding overhead and further assume that AdLeaks can recover a data transmission with probability at least 0.9 then submitting a 2 MB file requires about 1010 transmissions or 21 ± 1 days on average.

6.2 Network Load

A Base64 encoded ciphertext requires transmitting about 4496 bytes if we include 400 bytes worth of HTTP headers. At 50 transmissions a day this adds up to an additional daily network load of 220 KB/user. Since flat rates for Internet access are common in developed countries, we believe this is insignificant for users in these countries. Also, given that whistleblowers provide large societal benefits, society may well be willing to pay even a small, but significant, cost in bandwidth, beyond that needed by AdLeaks. Although one might get concerned that the accumulated load poses a problem at Internet scale. This is not likely the case. The quoted amount of data is less than the size of an average web page on the wire [31], which Google engineers found to be about 320KB in 2010. Hence, the load AdLeaks adds to the network is well within users' network traffic variance. It may not even be noticed against the backdrop of video streaming and increasing Internet use.

6.3 Guards and Aggregators

In order to establish a lower bound of the request rate that AdLeaks servers would be able to process we deployed guard and aggregator prototypes (see Section 8) on EmuLab [41]. We used six pc3000 nodes (3.0 GHz 64-bit Xeon processors, 2 GB DDR2 RAM, 1 Gbit connectivity) for guards, one d710 node (2.4 GHz 64-bit Quad Core Xeon E5530 processor, 12 GB RAM, 1 Gbit connectivity) for one aggregator and another pc3000 node for one decryptor. The guard servers sent chunks to the aggregator through a reverse SSH tunnel. The aggregator aggregated the incoming chunks and sent aggregates to the decryptor through a reverse SSH tunnel. The decryptor performed the tree decryption and discarded the decrypted chunks. Overload situations were easily observed through buffer growth, that is, the incoming rate exceeded the rate at which the aggregator processed and sent its aggregates. The aggregator operated stable at an incoming rate of 8500 chunks per second or roughly 209 Mb/s. Since our prototype does not yet implement the outer encryption we measured the overhead of elliptic curve based key exchanges separately. For a 256-bit key we measured 0.006632 ms with a standard deviation of 0.000171 ms (100 iterations). If we correct for this overhead then the aggregator handles 8046 reqs/s. Since guards merely strip some encoding from incoming requests and pass them on we did not measure them separately and instead assume that they perform similar to aggregators without outer encryption, that is, we assume

they handle 8000 reqs/s.

Web surfers are most active during certain time windows during the day. If active times are reasonably uniform across the population then they shift with the time zone, which spreads the active windows. This prompts the following assumptions, which may capture, for example, the situation in the United States: the active time of users is between 6pm and 1am, and they live in three adjacent time zones. This means, conservatively, that the 50 transmissions per user we assumed before concentrate within an 11 hours window. The U.S. have about 138 million broadband Internet users at the age of 18 years and older [35]. The expected load on guards is therefore about 174243 reqs/s. However, we are rather interested in the *peak* load, for example, the load that is not exceeded in 99% of all cases. Towards this end we model arrival times using a Poisson distribution. For the calculation of the peak load we use the fact that the Poisson distribution is very close to the normal distribution for a mean larger than about 20. It is well known that about 99.7% of normally distributed events are within 3 standard deviations from the mean. Since the variance of the Poisson distribution equals its mean we have that, in 99.85% of all cases, the load will be at most 175495 reqs/s. Using this as our basis we conclude that we need 22 guard units and 22 aggregators.

6.4 Data Recovery

By design, the decryptor scalability does not depend on the number of users but only on the number of whistleblowers. This is what allows us to upper bound the number of copies of the AdLeaks decryption key. Based on speed test reports on regional ISPs their download bandwidths range from 4 megabit per second (Mb/s) to 31 Mb/s for a household Internet connection. As a basis for subsequent estimation we use the median rounded to Mb, that is, we assume that decryptors can download ciphertexts from aggregators with 18 Mb/s. At 3072 bytes per ciphertext this translates to 768 aggregate ciphertexts per second.

We estimate the data recovery probability of AdLeaks under the following aggregation model. Given k gray ciphertexts and an arbitrary number of white ones, what is the probability that the data from a random gray ciphertext can be recovered if we can transmit m aggregates from aggregators to decryptors? The data of a gray ciphertext is recoverable if it is aggregated only with white ones. Therefore, we seek the probability that, given any gray ciphertext, all other gray ciphertexts fail to be assigned to the same aggregate, of which there are m. If k is small with respect to m/t for some t then we can improve our recovery probability as follows. We perform t independent aggregations with t sets of m/t aggregates.

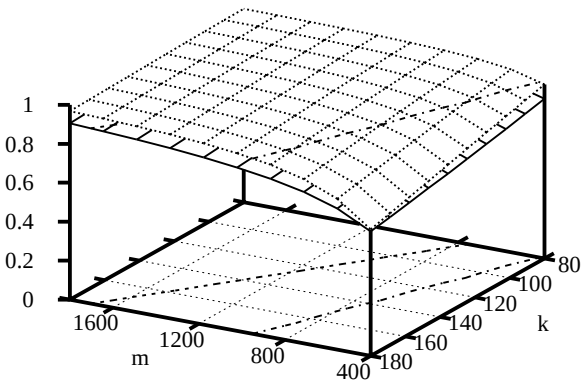

Figure 2: Shows the graphs of the recovery probability for varying numbers of data chunks and aggregates for $t = 1$ and $t = 4$. Contour lines indicate a 0.9 probability level.

This yields the same overall number m of aggregates. A gray ciphertext is recoverable if it is recoverable from any of the t sets. Hence, the probability we seek is one minus the probability that we fail to recover the data from all of the t sets of aggregates. Both probabilities are given by the following formulas:

$$\Pr[i = 1] = (1 - \frac{1}{m})^{k-1}$$
$$\Pr[i = 1] = 1 - (1 - (1 - \frac{1}{m/t})^{k-1})^t$$

Figure 2 illustrates the effect for $t = 1$ and for $t = 2$. The contour lines indicate where the recovery probability becomes larger than 0.9. For example, if aggregators can transfer 768 aggregates per second to a decryptor and whistleblowers send at most 82 gray ciphertexts per second and we set $t = 1$ then AdLeaks can recover each transmission with a probability of at least 0.9.

Next, we estimate the number of cryptographic operations AdLeaks must perform in order to decrypt with its tree decryption algorithm. Assume AdLeaks builds trees of depth n. This requires $2^n - 1$ modular multiplications. The expected number of decryptions is

$$E(X) = 1 + \frac{1}{2} \cdot \sum_{i=1}^{n} 2^i \cdot (1 - (1 - p)^{2^{n+1-i}})$$

Proof. The root of the tree must always be decrypted, hence the expectation is always at least 1. Since the algorithm only decrypts left children and not right children, we need to count only half of the remaining nodes. Recall that a right child is calculated by subtracting its

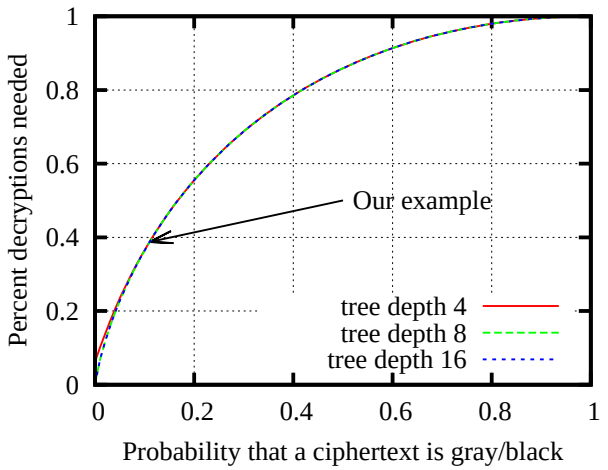

Figure 3: Shows the normalized savings of the tree decryption algorithm for various tree sizes and load characteristics. For $t > 8$ the graph looks identical to the graph for $t = 8$.

sibling from its parent. At level i from the root, starting under the root node, we have 2^i nodes. We have to decrypt a left child at level i if its parent is not zero. The probability that the parent is not zero is one minus the probability that its 2^{t+1-i} leaves are all zeroes. □

If we divide the expected number of decryptions by 2^n decryptions (the naïve approach) and plot the normalized results for several values of n then we obtain the graphs in Figure 3. The graphs tell us, for example, that we expect to save 0.61% of the decryptions if AdLeaks operates at its limit, that is, a recovery probability of 0.9 and $82/768 \approx 0.11\%$ gray or black ciphertexts. The lighter the load is the more we save.

6.5 Number of Whistleblowers

We characterize next how many concurrent whistleblowers a link capacity of 82 data chunks per second can support. Since no data set exists that we could use to estimate the arrival times of data chunks, we model them as a Poisson process that we approximate by a normally distributed process as before. To be on the safe side we seek a safe average sending rate r so that the actual sending rate does not exceed 82 reqs/s in 97.725% of all cases, that is, the second quantile. The safe average rate can be found easily by solving $r + 2 \cdot \sqrt{r} = 82$ for r, which yields 65 reqs/s. At 50 transmissions per day and whistleblower this means that AdLeaks can serve 51480 concurrent whistleblowers at any time with a 18 Mb/s uplink for the decryptor.

6.6 Decryptors

The performance of the decryptor is bound by the cost of two operations: the time it takes to test whether a tree node is white, and the time it takes to perform a full decryption and verification. We measured 0.011578 and 0.192843 seconds for these operations on a single core (2.66GHz Intel Xeon) with negligible standard deviation. If we assume that the decryptor has 11 cores available for decryption then we estimate that our running example requires 2 decryptors. Since the necessary resources for decryption scale linearly with the number of whistleblowers this means that AdLeaks can serve about 25740 concurrent whistleblowers with just one unit, for example, a dual 6-core Mac Pro.

6.7 Client Measurements

Our JavaScript implementations of the DJ scheme leverage several optimizations [23] that improve efficiency over unoptimized DJ by a factor of 8 to 32 in our measurements. As a side effect, the bit lengths of two parameters of our inner encryption scheme, namely r_1, r_2 (see Section 4), bound the time it takes to encrypt in the browser. Reasonable values range from 512 bits (probably sufficient) to 2044 bits (paranoid). We measured these times on an Intel i5-2500K CPU at 3.30 GHz with Chromium Version 20.0.1132.47. For 512 bits it took 7.55 seconds ($s = 0.06$, speedup ≈ 32), for 1024 bits it took 14.67 seconds ($s = 0.05$, speedup ≈ 16) and for 2044 bits it took 28.68 seconds ($s = 0.38$, speedup ≈ 8). Since Java is still significantly faster than JavaScript we assume that these times will become smaller still in the future.

7 Financial Viability

Given the server resources AdLeaks requires it is prudent to ask what are the costs the AdLeaks organization has to bear. We found that dual quad-core servers with unmetered 1Gb interfaces are available for less then 400 USD/month. At this price, the infrastructure for the guards and aggregators necessary to serve 138 million users would cost 579 USD/day. While this seems high it is instructive to look at the revenue side. Each ad loading corresponds to what is called an "impression" in the online advertising business. The prices for impressions are typically quoted as *costs per mille,* or CPM. Top ranking websites can command prices over 100 USD while run-off-the-mill websites receive in the order of 0.25 USD. *Cost per click* models or *cost per conversion* models generate additional revenue, which we ignore for the sake of conservatism. If AdLeaks had 138 million users who see 5 AdLeaks ads per day on average and if AdLeaks paid

out 0.25 USD per mille impressions and if its markup was 0.34 percent or better then AdLeaks would break even on its infrastructure costs. For comparison, ValueClick Inc. reported a gross profit of over 96 million USD on a revenue of about 161 million USD in its second quarter of 2012, which translates to about 59 percent profitability, and reported 25 cents of net income per common share. The numbers suggest that, if AdLeaks was run as a not-for-profit operation, it could gain market share by offering very competitive pricing while earning a decent plus. This is in sharp contrast to contemporary whistleblowing platforms who depend entirely on donations.

8 Implementation

We developed fully-functional multi-threaded aggregation and decryption servers with tree decryption support as well as a Fountain Code encoder and decoder. Decryptors write recovered data to disk and the decoder recovers the original file. We also developed a fake guard server which is capable of generating and sending chunks according to a configurable ratio of white and gray ciphertexts. All servers connect to each other through SSH tunnels via port forwarding. The entire implementation consists of 101 C, header and CMake files with 7493 lines of code overall. This includes our optimized DJ implementation [23], which is based on a library by Andreas Steffen, a SHA-256 implementation by Olivier Gay, and several benchmarking tools. Our ads implement the DJ scheme based on the *JSBN.js* library and use *Web Workers* to isolate the code from the rest of the browser. The entire ad currently measures less than 81 KB. The size can be reduced further by eliminating unused library code and by compressing it. The ad submits ciphertexts via *XmlHttpRequests.* We instrumented the Firefox browser for our prototype and patched the source code in two locations. First, we hook the compilation of *Web Worker* scripts and tag every script as an AdLeaks script if it is labeled as one in lieu of carrying a valid signature. We placed a second hook where Firefox implements the *XmlHttpRequest.* Whenever the calling script is an AdLeaks script running within a *Web Worker,* we replace the zero chunk in its request with a data chunk.

9 Related Work

Closely related to our work, there is early work on DC Nets [14, 39] which aims to provide a cryptographic means to hide who sends messages, the use of Raptor codes [10] to implement an asynchronous unidirectional one-to-one and one-to-many covert channel using spam messages, and anonymous data aggregation [30] for distributed sensing and diagnostics. In preserving the privacy of web-based email [7] one can take advantage of a spread-spectrum approach for hiding the existence of a message, but it is not secure against a global attacker. Membership-concealment [36] can also be used to hide the real-world identities of participants in an overlay network, but this doesn't suffice for AdLeaks.

In censorship resistance, there is Publius [40] which is an anonymous publication system but does not offer any sort of connection-based anonymity. Collage [6] steganographically embeds content in cover traffic such as photo-sharing sites and implements a rendezvous mechanism to allow parties to publish and retrieve messages in this cover traffic, but Alice and Bob must exchange a key a priori. More recent work [22] explores an approach that assumes the ability of being able to globally check and retrieve all blog posts in real time and determining and extracting all the embedded content.

Another related area is secure data aggregation in wireless sensor networks [2] or WSNs. One can try to securely aggregate encrypted data [9], which identifies the key stream in the header and requires removing a stream for each ciphertext received. The latter wouldn't scale for AdLeaks because it requires millions of keystream removals per aggregate. One approach [37] for aggregating in multicast communication uses the Okamoto-Uchiyama encryption scheme for secure aggregation, which resembles Pederson's commitment scheme very closely. The difference is really that AdLeaks is not used in the aggregate but instead deals with collisions. Other work [38, 13] targets the security of statistical computations on the inputs from various sensor nodes. The two key differences in the approaches found in WSNs are: (i) WSNs want correctly aggregated data that allows for unencrypted sending, whereas our approach seeks the opposite of that, and (ii) the attacker wants to have a tainted input accepted, whereas in our context he wants to learn the content of the input. In WSNs both event-driven and query-based processing is of interest, with most approaches focusing on query-based solutions, whereas AdLeaks is event-driven. That also means that we don't know *a priori* which client sends what in what round. It is difficult for us to exchange keys beforehand and our approach remains unidirectional, i.e. we cannot distribute keys. Lastly, in WSNs clients are not trusted initially and are later vetted, whereas in our approach clients are never trusted.

10 Conclusions

AdLeaks leverages the ubiquity of online advertising to provide anonymity and unobservability to whistleblowers making a disclosure online. The system introduces

a large amount of cover traffic in which to hide whistleblower submissions, and aggregation protocols that enable the system to manage the huge amount of traffic involved, enabling a small number of trusted nodes with access to the decryption keys to recover whistleblowers' submissions with high probability. We analyzed the performance characteristics of our system extensively. Our research prototype demonstrates the feasibility of such a system. We expect many aspects of the system can be improved and optimized, providing ample opportunity for further research.

Acknowledgements

The first, second and last author are supported by an endowment of *Bundesdruckerei GmbH*. An abridged version of this paper has been accepted for publication in the proceedings of *Financial Cryptography and Data Security 2013* [33]. Copies of the abridged version will eventually become available at `http://www.springer.de/comp/lncs/`.

References

[1] Alexa. Online at `http://www.alexa.com`, Apr. 2012.

[2] H. Alzaid. *Secure Data Aggregation in Wireless Sensor Networks*. PhD thesis, Queensland University of Technology, March 2011.

[3] D. Banisar. *Whistleblowing — International Standards and Developments*. Transparency International, Feb. 2009.

[4] S. Berthold, R. Böhme, and S. Köpsell. Data retention and anonymity services. In *The Future of Identity in the Information Society*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 92–106. Springer, 2009.

[5] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo. Baiting inside attackers using decoy documents. In *Proc. SecureComm*, pages 51–70. Springer, 2009.

[6] S. Burnett, N. Feamster, and S. Vempala. Chipping away at censorship firewalls with user-generated content. In *Proc. USENIX Security*, pages 29–29, 2010.

[7] K. Butler, W. Enck, J. Plasterr, P. Traynor, and P. McDaniel. Privacy preserving web-based email. In *Proc. International Conference on Information Systems Security*, ICISS, pages 116–131. Springer, 2006.

[8] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing chosen-ciphertext security. In *Proc. CRYPTO*, volume 2729 of *LNCS*, pages 565–582. Springer, 2003.

[9] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik. Efficient and provably secure aggregation of encrypted data in wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(3):20:1–20:36, June 2009.

[10] A. Castiglione, A. De Santis, U. Fiore, and F. Palmieri. An asynchronous covert channel using spam. *Comput. Math. Appl.*, 63(2):437–447, Jan. 2012.

[11] E. R. Center. 2011 National Business Ethics Survey. Online at `http://www.ethics.org/nbes`, 2345 Crystal Drive, Suite 201, Arlington, VA 22202, USA, 2012.

[12] S. Chakravarty, A. Stavrou, and A. D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *Proc. ESORICS*, pages 249–267. Springer, 2010.

[13] H. Chan, A. Perrig, B. Przydatek, and D. Song. Sia: Secure information aggregation in sensor networks. *J. Computer Security*, 15(1):69–102, Jan. 2007.

[14] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1:65–75, 1988. 10.1007/BF00206326.

[15] D. Dahl and R. Sleevi. Web cryptography API. W3C Draft, October 2012.

[16] I. Damgård, M. Jurik, and J. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9:371–385, 2010.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *Proc. USENIX Security Symposium*, pages 303–320, 2004.

[18] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[19] A. Freier, P. Karlton, and P. Kocher. The secure sockets layer (SSL) protocol version 3.0. RFC 6101 (Historic), Aug. 2011.

[20] S. Gustin. Columbia university reverses anti-WikiLeaks guidance. `http://www.wired.com/threatlevel/2010/12/columbia-wikileaks-policy/`, Dec. 2010.

[21] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proc. ACM CCS*, 2011.

[22] L. Invernizzi, C. Kruegel, and G. Vigna. Message In A Bottle: Sailing Past Censorship. In *Hot Topics in Privacy Enhancing Technologies*, Vigo, Spain, 2012.

[23] M. J. Jurik. *Extensions to the Pailler Cryptosystem with Applications to Cryptological Protocols*. Dissertation, BRICS, Department for Computer Science, University of Aarhus, Aug. 2003.

[24] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable internet communication. In *Proc. Workshop on Free and Open Communications on the Internet*. USENIX, 2011.

[25] A. Klein. Temporary user tracking in major browsers and cross-domain information leakage and attacks. Technical report, Trusteer, 2008. Online at `http://www.trusteer.com`.

[26] K. J. Lennane. "Whisteblowing": a health issue. *British Medical Journal*, 307:667–670, Sept. 1993.

[27] D. MacKay. Fountain codes. *IEE Proceedings*, 152(6), Dec. 2005.

[28] A. Osterhaus and C. Fagan. *Alternative to Silence — Whistleblower Protection in 10 European Countries*. Transparency International, 2009.

[29] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO*, volume 576, pages 129–140. Springer, 1992.

[30] K. P. N. Puttaswamy, R. Bhagwan, and V. N. Padmanabhan. Anonygator: privacy and integrity preserving data aggregation. In *Proc. ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware, pages 85–106. Springer, 2010.

[31] S. Ramachandran. Web metrics: Size and number of resources. Online at `https://developers.google.com/speed/articles/web-metrics`, May 2010.

[32] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6:365–403, Aug. 2003.

[33] V. Roth, B. Güldenring, E. Rieffel, S. Dietrich, and L. Ries. A secure submission system for online whistleblowing platforms. In *Proc. Financial Cryptography and Data Security*, LNCS. Springer Verlag, 2013. To appear.

[34] E. Stark, M. Hamburg, and D. Boneh. Symmetric cryptography in javascript. In *Proc. ACSAC*, pages 373–381. IEEE Computer Society, 2009.

[35] Persons using the Internet in and outside the home. `http://www.ntia.doc.gov/files/ntia/data/CPS2010Tables/t11_1.txt`, Jan. 2011.

[36] E. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim. Membership-concealing overlay networks. In *Proc. ACM CCS*, pages 390–399, 2009.

[37] A. Viejo, Q. Wu, and J. Domingo-Ferrer. Asymmetric homomorphisms for secure aggregation in heterogeneous scenarios. *Inf. Fusion*, 13(4):285–295, Oct. 2012.

[38] D. Wagner. Resilient aggregation in sensor networks. In *Proc. ACM Workshop on Security of Ad Hoc and Sensor Networks*, SASN, pages 78–87. ACM, 2004.

[39] M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *EUROCRYPT*, volume 434 of *LNCS*, pages 690–690. Springer, 1990.

[40] M. Waldman, A. Rubin, and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proc. USENIX Security Symposium*, pages 59–72, Aug. 2000.

[41] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. OSDI*, pages 255–270, Dec. 2002.

[42] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proc. USENIX Security Symposium*, Aug. 2011.

[43] H. Zhang and S. Zhao. Measuring web page revisitation in tabbed browsing. In *Proc. CHI*, pages 1831–1834, 2011.